

알고리즘 입문1

1. big-Oh : 점근적 상한

$$O(f(n)) = \{g(n) \mid \exists c > 0, n_0 \geq 0 \text{ s.t. } \forall n \geq n_0, cf(n) \geq g(n)\}$$

$\Rightarrow n \geq n_0 \rightarrow cf(n) \leq g(n)$ 를 만족하는 C 의 존재를 보이면 증명

2. omega : 점근적 하한

$$\Omega(f(n)) = \{g(n) \mid \exists c > 0, n_0 \geq 0 \text{ s.t. } \forall n \geq n_0, cf(n) \leq g(n)\}$$

3. Theta

$$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$$

점화식의 점근적 분석 방법

1. 반복대치 2. 추정 후 증명 3. 마스터정리

2. 귀납적 증명

i) $n=1$ $T(n) \leq Cf(n)$ 확인

ii) $n=k$ $T(k) \leq Cf(k)$ 이라고 가정할 때

$n=k+1$ $T(k+1) \leq Cf(k+1)$ 을 만족함을 보인다.

3. 마스터 정리

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$n^{\log_b a} = h(n)$$

$$\textcircled{1} \quad \frac{f(n)}{h(n)} = \frac{1}{n^\epsilon} \rightarrow T(n) = \Theta(h(n))$$

$$\textcircled{2} \quad \frac{f(n)}{h(n)} = n^\epsilon, \quad af\left(\frac{n}{b}\right) \leq cf(n) \text{ 이면 } T(n) = \Theta(f(n))$$

$$\textcircled{3} \quad \frac{f(n)}{h(n)} = \Theta(1) \quad T(n) = \Theta(h(n) \log n)$$

heap sort \Rightarrow make heap + heapify

make heap() {

$i \rightarrow \frac{n}{2} \text{ to } 1$

{ heapify(i) }

}

heapify() $\rightarrow \log n$

$$\log n \times (n-1) = O(n \log n)$$

$\overbrace{\log \frac{n}{2} + \dots + \log 2 + \log 1}^{n/2 \text{ 번 } \log k \text{ 실행}}$

$$\Rightarrow \log \frac{n}{2} + \dots + \log 2 + \log 1 = \Theta(n)$$

$$\leq \log \frac{n}{2} + \dots + \log \frac{n}{2} + \log \frac{n}{2}$$

$$\frac{n}{2} \log \frac{n}{2}$$

최악 선형 선택 알고리즘

linearSelection(int A[], int P, int r, int i) // i번째 원소 찾기

- ① 5개 이하이면 탐색해서 찾는다. n
- ② $\lceil \frac{n}{5} \rceil$ 개의 그룹으로 나눈다. n
- ③ 각 그룹에서 중앙값을 찾는다. $\frac{n}{5} \times 5 = n$
- ④ 중앙값들의 중앙값을 재귀적으로 구한다. linearSelection(M[], 0, n/10)
- ⑤ M을 기준으로 분할 n (M의 인덱스 확인)
- ⑥ 적합한 그룹으로 선택해 선택 재귀 $n - (\frac{n}{5} \times \frac{1}{2} \times 3 - 3 + 1) = \frac{7}{10}n + 2$

?

⇒ ① If((r-P+1) <= 5) → Sort(A, P, r)

If((P+k) == i) → return A[P+k];

② Sort(A, P, K₁), ..., Sort(A, K_[$\frac{n}{5}$], r)

③ for(j=P+2; j<r; j+=5) → arr[j] = A[j]

int arr[] = {m₁, ..., m_[$\frac{n}{5}$]}

④ int M = linearSelection(arr, 0, [n/5]/2)

⑤ int index = Partition(A, M)

⑥ If(i < index) linearSelection(A, P, index-1, i)

else linearSelection(A, index, r, T)

추정 후 증명

$$T(n) \leq T(n/5) + T(7n/10 + 2) + \Theta(n)$$

$$\leq \underline{T(n/5+1)} + \underline{T(7n/10+2)} + \Theta(n)$$

$T(n) \leq cn$ 라고 가정

$$\leq c(n/5+1) + c(7n/10+2) + \Theta(n)$$

$$\leq \frac{9c}{10}n + 3c + \Theta(n)$$

$$\leq cn - \frac{1}{10}cn + 3c + \Theta(n)$$

$$\leq cn \quad (n \geq n_0 \rightarrow c \text{가 존재})$$

RedBlack Tree Rule

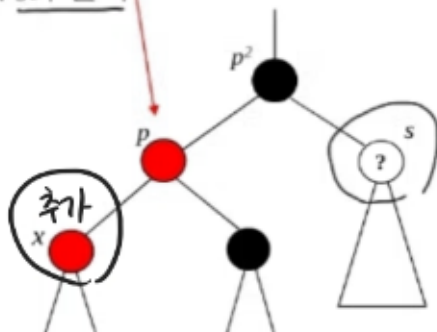
1. 루트는 블랙이다.
2. 모든 리프는 블랙이다.
3. 노드가 레드이면 그 노드의 자식은 반드시 블랙이다
4. 루트 노드에서 임의의 리프노드에 이르는 경로에서 만나는 블랙 노드의 수는 모두 같다.

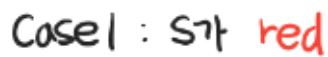
Insert (삽입 시 노드는 red)

P가 red인 경우 error!!

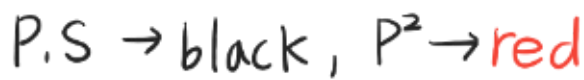
Case 1: s가 레드

Case 2: s가 블랙





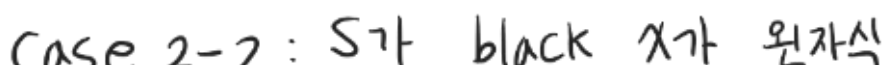
Case 1 : S7가 red

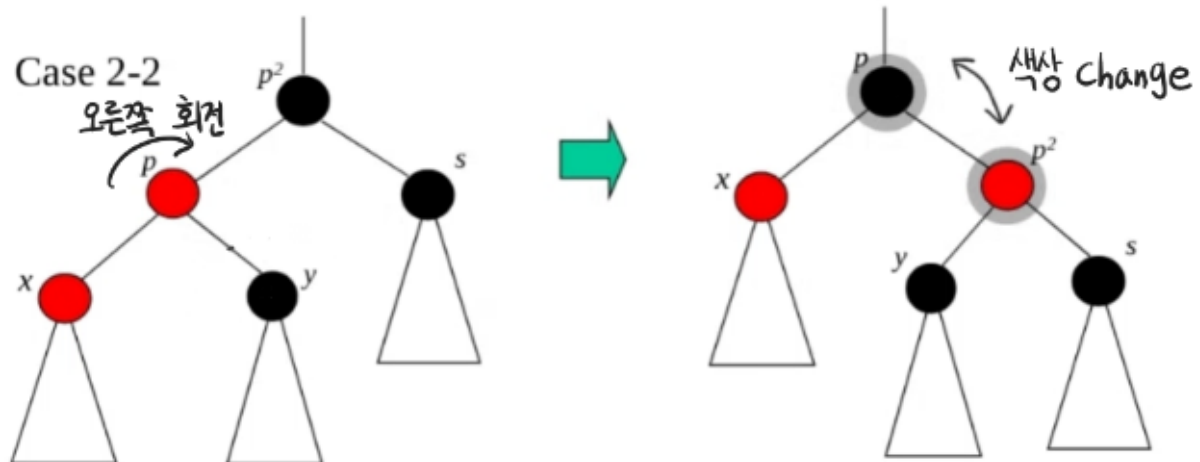


- P^2 가 루트면 $P^2 \rightarrow \text{black}$ 끝!
- P^2 가 루트가 아니면 P^2 의 부모 색상을 확인
 $\hookrightarrow P^3$ 가 black이면 끝!

P^3 가 red이면 $(\underset{r}{P^2}, \underset{r}{P^3}) \rightarrow (\underset{r}{x}, \underset{r}{P})$ 로 재귀

CASE 2-1 : S가 black 자가 모든 자식





```
#include "redblack.h"
```

```
IntRBTREE::IntRBTREE() {
    z = new Node(black, 0, 0, 0, 0);
    z->l = z; z->r = z;
    head = new Node(black, 0, 0, 0, z);
}
```

leaf 노드는
코를 가르켜고 있다.

```
Node *IntRBTREE::search(int search_key)
{
    Node *x = head->r;
    while (x != z) {
        if (x->key == search_key) return x;
        x = (x->key > search_key) ? x->l : x->r;
    }
    return NULL;
}
```

```
void IntRBTREE::insert(int v, int index)
{
    x = head; p = head; g = head;
    while (x != z) {
        gg = g; g = p; p = x;
        if (x->key == v) return;
        x = (x->key > v) ? x->l : x->r;
        if (x->l->b && x->r->b) this->split(v);
    }
    x = new Node(red, v, index, z, z);
    if (p->key > v) p->l = x;
    else p->r = x;
    this->split(v); head->r->b = black;
}
```

leaf로 가기

x의 자식들다

split

P 밑에 Insert

root black으로

P에 달고 split

```

void IntrBTree::split(int v)
{
    x는 red로 자식은 black으로
    x->b = red; x->l->b = black; x->r->b = black;
    if (p->b) { p도 red이면
        g->b = red; g를 red로 한 뒤
        if (g->key > v != p->key > v) p = this->rotate
            (v, g);
        x = this->rotate(v, gg);
        x->b = black;
    }
}

```

Case 2-1

Case 2-2

```

Node *IntrBTree::rotate(int v, Node *y)
{
    y를 중심으로 rotate
    Node *gc, *c; (올라온 gc를 출력)
    c = (y->key > v) ? y->l : y->r;
    if (c->key > v) {
        gc = c->l; c->l = gc->r; gc->r = c;
    }
    else {
        gc = c->r; c->r = gc->l; gc->l = c;
    }
    if (y->key > v) y->l = gc;
    else y->r = gc;
    return gc;
}

```

```

#ifndef REDBLACK_H
#define REDBLACK_H

const int black = 0;
const int red = 1;

#define MAX_STRKEY_LEN 20

class Node {
public:
    Node(int bb, int k, int i, Node *ll, Node *rr)
    {
        b = bb, key = k; index = i, l = ll, r = rr;
    };
    int b; // color
    int key; // data
};

```

②return

```
    int index;  
    Node *l, *r;  
};  
  
class IntRBTTree {  
public:  
    IntRBTTree();  
    Node *search(int search_key);  
    void insert(int v, int index);  
    void split(int v);  
    Node *rotate(int v, Node *y);  
private:  
    Node *head, *z, *gg, *g, *p, *x;  
};  
  
#endif
```



BST delete

treeDelete(t, r, P) {

 left[P] = deleteNode(r);


} t: root r: remove P: Parent

delete(r) {

 Case 1  return NIL;

 Case 2-1  return right[r];

 Case 2-2  return left[r];

 Case 3 

 S ← right[r]

 while (left[S] ≠ NIL) { Parent ← S; S ← left[S]; }

 if (S == right[r]) { right[r] ← right[S]; }

 else left[Parent] ← right[S];



return r;

}

B-tree

- root를 제외하고 $\lfloor \frac{k}{2} \rfloor \sim k$ 개의 키
- 모든 리프 노드는 같은 길이를 가진다.

BTreeInsert(t, x)

{

x를 삽입할 리프 노드 r을 찾는다;
x를 r에 삽입한다;

if (r에 오버플로우 발생) then clearOverflow(r);

}

clearOverflow(r)

{

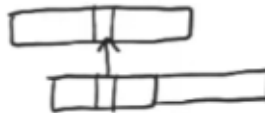
if (r의 형제 노드 중 여유가 있는 노드가 있음) then {r의 남은 키를 넘긴다};
else {

r을 둘로 분할하고 가운데 키를 부모 노드 p로 넘긴다;

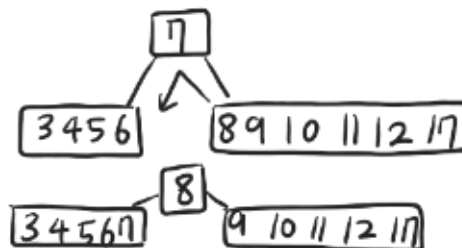
if (부모 노드 p에 오버플로우 발생) then clearOverflow(p);

}

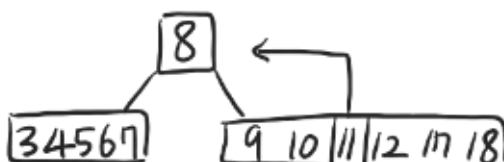
}



재분배



분할

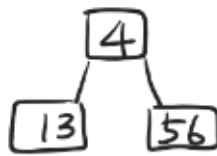


BTreeDelete(t, x, v)

```
{
  if (v가 리프 노드 아님) then {
    x의 직후원소 y를 가진 리프 노드를 찾는다;
    x와 y를 맞바꾼다;
  }
  리프 노드에서 x를 제거하고 이 리프 노드를 r이라 한다;
  if (r에서 언더플로우 발생) then clearUnderflow(r);
}
clearUnderflow(r)
{
  if (r의 형제 노드 중 키를 하나 내놓을 수 있는 여분을 가진 노드가 있음)
    then { r이 키를 넘겨받는다; }
  else {
    r의 형제 노드와 r을 합병한다;
    if (부모 노드 p에 언더플로우 발생) then clearUnderflow(p);
  }
}
```

▷ t : 트리의 루트 노드
 ▷ x : 삭제하고자 하는 키
 ▷ v : x 를 갖고 있는 노드

직후 원소와 교환 후 삭제 필수
 재분배



병합

